AFRL-DE-PS-
TP-2006-1019

AFRL-DE-PS-
TP-2006-1019

Parallelization of a blind deconvolution algorithm (Postprint)

Charles Matson
Kathy J. Borelli

1 September, 2006

Conference Proceeding

AIR FORCE RESEARCH LABORATORY
Directed Energy Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 1 September 2006	**2. REPORT TYPE** Conference Proceeding (Postprint)	**3. DATES COVERED** *(From - To)* Dec 5, 01- Sep 30, 06

4. TITLE AND SUBTITLE Parallelization of a blind deconvolution algorithm (Postprint)	**5a. CONTRACT NUMBER** F29601-01-D-0083 TO 6
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 69120G
6. AUTHOR(S) Charles L. Matson, Kathy J. Borelli	**5d. PROJECT NUMBER** 2304
	5e. TASK NUMBER B3
	5f. WORK UNIT NUMBER AA
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFRL/DESA 3550 Aberdeen AV SE Kirtland AFB, NM 87117-5776	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 3550 Aberdeen AV SE Kirtland AFB, NM 87117-5776	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/DESA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-DE-PS-TP-2006-1019

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited.

13. SUPPLEMENTARY NOTES

Author final manuscript published in proceedings of the High-Performance Computing Modernization Office Users Group Conference, published by IEEE, 2006. One or more of the authors is a U.S. Government employee working within the scope of their position; therefore, the U.S. Government is joint owner of the work and has the right to copy, Distribute, and use the work. Any other form of use is subject to copyright restrictions.

14. ABSTRACT

Often it is of inters to deblur imagery in order to obtain higher-resolution images. Deblurring requires knowledge of the blurring function- information that is often not available separately from the blurred imagery. Blind deconvolution algorithms overcome this problem by jointly estimating both the high-resolution image and the blurring function from the blurred imagery. Because blind deconvolution algorithms are iterative in nature, they can take minutes to days to deblur an image depending how many frames of date are used for the deblurring and the platforms on which the algorithms are executed. Here we present our progress in parallelizing a blind deconvolution algorithm to increase its executions speed. This progress includes sub-frame parallelization and a code structure that is not specialized to a specific computer hardware architecture.

15. SUBJECT TERMS

Multi-frame blind deconvolution, blind deconvolution, high-performance computing, parallel processing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Charles Matson
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	SAR	10	**19b. TELEPHONE NUMBER** *(include area code)* N/A

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

Parallelization of a blind deconvolution algorithm

Charles L. Matson[a] and Kathy J. Borelli[b]

[a]Optics Division, AFRL/DES, Directed Energy Directorate
U.S. Air Force Research Laboratory, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776 USA
[b]KJS Consulting, 71 Awalau Road, Haiku, HI, 96706 USA

ABSTRACT

Often it is of interest to deblur imagery in order to obtain higher-resolution images. Deblurring requires knowledge of the blurring function – information that is often not available separately from the blurred imagery. Blind deconvolution algorithms overcome this problem by jointly estimating both the high-resolution image and the blurring function from the blurred imagery. Because blind deconvolution algorithms are iterative in nature, they can take minutes to days to deblur an image depending how many frames of data are used for the deblurring and the platforms on which the algorithms are executed. Here we present our progress in parallelizing a blind deconvolution algorithm to increase its execution speed. This progress includes sub-frame parallelization and a code structure that is not specialized to a specific computer hardware architecture.

Keywords: Multi-frame blind deconvolution, blind deconvolution, high-performance computing, parallel processing

1. INTRODUCTION

Any image obtained using an optical system is blurred by the optical system itself and may be further blurred by other mechanisms such as image motion, unstable imaging system platforms, and atmospheric turbulence. If the spatial distribution of the blurring function is known, the image can be deblurred using standard deconvolution methods. However, often the blurring function is not known. In this case, blind deconvolution algorithms can be used to jointly estimate the deblurred image and the blurring function from the blurred image. In many cases, there is a sequence of blurred images that consists of a pristine image that is common to all the blurred images, while the blurring is different for each image in the sequence. Multi-frame blind deconvolution (MFBD) algorithms are used to jointly estimate the deblurred image common to all the images as well as all the separate blurring functions.[1] In this paper, we will make no further distinction between blind deconvolution and MFBD algorithms, since MFBD algorithms can be used with a single image as well as sequences of images.

Although MFBD algorithms are robust algorithms that produce high-quality deblurred images with a minimum of a priori knowledge, they are iterative in nature and thus relatively slow. Depending upon the sizes of the images, the number of images included in the estimation process, and the computing platform, serial versions of MFBD can take minutes to days to generate a single deblurred image. For this reason, we have been conducting research into parallelizing our implementation of an MFBD algorithm called Physically-Constrained Iterative Deconvolution (PCID). A key emphasize in the research presented herein is demonstrating the scalability of PCID across multiple processors in a distributed-memory environment. In this paper we report on our progress. The outline of the paper is as follows: in Section 2 we describe the PCID algorithm with an emphasis on its past and current levels of parallelization, in Section 3 we discuss the computational environments for which PCID has been and is being parallelized, in Section 4 we present results showing the speed and scalability of the current level of parallelization, and in Section 5 we give conclusions and planned future work.

2. THE PCID ALGORITHM

An overview of the PCID algorithm is first given in this section. Next, we give the mathematics of the PCID algorithm. Finally, issues associated with parallelizing the PCID algorithm for various levels of parallelization are presented and discussed.

AFRL/DE 06-390

2.1 PCID overview

The PCID algorithm is a multi-frame blind deconvolution algorithm that jointly estimates the pristine image and the blurring functions that generated the sequence of blurred images used as input to the algorithm. The estimates of the pristine image and blurring functions are obtained by minimizing a cost function that consists of two terms. The first term is a data-matching term that is the weighted sum (over image pixels and measurement frames) of the measurement/model residuals. The measurement/model residuals are defined as the differences between the actual measurements and the current models of the measurements, where the current models of the measurements are generated from the current estimates of the pristine image and blurring functions by convolving each blurring function with the pristine image. The residuals are weighted by the noise covariance matrix, assumed diagonal for computational tractability. The second term in the cost function is a regularization term that prevents excessive amplification of the measurement noise that occurs when the data-matching term is minimized without constraint. A scaling parameter multiplies the regularization term and is chosen to optimally trade off the resolution with the noise levels in the deblurred image.

2.2 PCID mathematics

Let $i_k(x)$, $k=1,\ldots,N_{tot}$, be a sequence of N_{tot} images that have a common pristine image $o(x)$ and N_{tot} different blurring functions $h_k(x)$, $k=1,\ldots,N_{tot}$, where x is a two-dimensional vector specifying a location in image space. We denote the collection of the N_{tot} blurring functions by $\{h_k(x)\}$. The kth image $i_k(x)$ is the sum of the convolution of $o(x)$ with $h_k(x)$ and a noise term:

$$i_k(x)=o(x)*h_k(x)+n_k(x) \tag{1}$$

where the asterisk denotes the convolution operation and $n_k(x)$ is the zero-mean noise, with variance $\sigma_k^2(x)$, associated with the kth image. The PCID algorithm finds the estimate $\hat{o}(x)$ of $o(x)$ and the estimates $\{\hat{h}_k(x)\}$ of $\{h_k(x)\}$ that minimize the cost $J\left[\hat{o}(x),\{\hat{h}_k(x)\}\right]$ defined by

$$J\left[\hat{o}(x),\{\hat{h}_k(x)\}\right]=\sum_{k=1}^{N_{tot}}\sum_{n=1}^{N_i^2}\frac{1}{\sigma_k^2(x_n)}\left[i_k(x_n)-\hat{i}_k(x_n)\right]^2+\lambda\sum_{n=1}^{N_i^2}\hat{o}^2(x_n) \tag{2}$$

where $\hat{i}_k(x)=\hat{o}(x)*\hat{h}_k(x)$ is the model of the kth measurement, N_i is the linear dimension of the arrays containing the object, the blurring functions, and the images, x_n is the nth pixel in each of these arrays, and λ is a real constant. The minimization of Eq.(2) is carried out with respect to the parameters that generate $\hat{o}(x)$ and $\{\hat{h}_k(x)\}$; therefore, to carry out the minimization, the appropriate parameters must be chosen. We choose to generate $\hat{o}(x)$ pixel by pixel for maximum flexibility. However, the PCID algorithm has available two separate parameter sets that can be used to generate $\{\hat{h}_k(x)\}$. The first parameter set is the set of pixel intensities, just as for $\hat{o}(x)$. The second parameter set exploits the fact that we know that $\{\hat{h}_k(x)\}$ are blurring functions produced by an optical system (potentially with phase aberrations in the pupil). Taking advantage of this prior knowledge, we create $\{\hat{h}_k(x)\}$ using a Zernike polynomial expansion of the phase in the pupil incorporated in a Fourier optics model.[2] The parameter set in this case is the set of the coefficients of the Zernike polynomials. For atmospherically-corrupted images, we have found that using ~100 Zernike polynomials in the expansion produces good results. The Zernike-based parameterization of $\{\hat{h}_k(x)\}$ tends to produce lower-noise and higher-resolution estimates of $o(x)$ as compared to the pixel parameterization of $\{\hat{h}_k(x)\}$, due both to the inclusion of the Fourier optics model in the generation of $\{\hat{h}_k(x)\}$ and the reduced degrees of freedom (~100 Zernike coefficients versus ~16,000 pixel intensities for a 128 by 128 image).

Once the parameter sets are chosen, we use the Numerical Recipes' conjugate gradient routine,[3] modified to execute in parallel, to carry out the minimization of Eq.(2) with respect to these parameters. This routine requires the user to provide subprograms that return, for the current estimates $\hat{o}(\mathbf{x})$ and $\{\hat{h}_k(x)\}$, the value of the cost as well as the gradient of the cost with respect to an arbitrary parameter for either $\hat{o}(\mathbf{x})$ or $\{\hat{h}_k(x)\}$. The conjugate gradient routine seeks to find the global minimum of the cost by carrying out searches for minima along lines in conjugate directions. Each iteration in the PCID algorithm corresponds to a conjugate direction. It is assumed that the desired solution has been reached when the change in the cost from one iteration to the next is less than a specified minimum value.

For the estimation problem to be unique; i.e., for there to be a unique o(x) and unique $\{h_k(x)\}$ given the measurements $\{i_k(x)\}$, additional information must be included in the PCID algorithm. Two common types of information are the knowledge that the pixel intensities of o(x) are non-negative and that o(x) is zero outside a region known as the support of o(x). In addition, we have had no noticeable problems with generating unique solutions when $\{\hat{h}_k(x)\}$ is parameterized using a Zernike-polynomial decomposition of the phase in the pupil, as is not the case for a pixel-parameterization of $\{\hat{h}_k(x)\}$. For that reason, along with the improved image reconstruction quality mentioned above, we routinely use the Zernike-coefficient parameterization of $\{\hat{h}_k(x)\}$ and not the pixel parameterization. We comment in passing that, of course, it is impossible to estimate frequency components of o(x) that are outside the bandpass of the optical system (excluding superresolution methods that are not considered here); thus, when we say that there is a unique estimate $\hat{o}(\mathbf{x})$ of o(x) generated by PCID, we are using the term 'unique' in the context of the regularization enforced by the second summation in Eq.(2).

2.3 PCID parallelization

We have chosen to use a single master process – multiple worker processes paradigm under which to parallelize PCID. The master process program and the worker process program are separate programs that run in the MPICH parallel processing environment.[4] The master process first reads in the image sequence, carries out initialization steps, and farms out images from the image sequence to the worker processes. After this startup sequence, the master process uses results from calculations provided by the worker processes to generate conjugate directions, find minima in each conjugate direction, and evaluate the cost to see when the minimum cost has been achieved.

The degree of parallelization in PCID is determined by the amount of processing carried out by the worker processes, how many worker processes are used, and the number of processors in each worker process. Prior to the research described in this paper, we had parallelized PCID to the extent of being able to use a single worker process per image in the image sequence, where each worker process uses only one processor. These worker processes generate the portions of the cost and the gradient of the cost corresponding to the images given to them. The master process collects these portions from all the worker processes, combines them to generate the complete cost and gradient of the cost, and carries out all other calculations needed to minimize the cost. There is no communication or data transfer necessary between worker processors because there is only one worker processor per worker process. At this level of parallelization, execution time is a monotonically-decreasing function of the number of processors used by PCID for at least up to tens of worker processes. Of course, the maximum number of processors that can be used is equal to the number of images used in the reconstruction plus one for the master process. Because it is often the case that the number of images in an image sequence is much less than the number of available processors, additional parallelization is desirable.

Since this previous work, we have further increased the level of parallelization in PCID in two ways. The first way is by carrying out all possible calculations using the worker processes that use the data that each worker process already has in local memory. For example, the worker processes are now employed to calculate quantities used to generate a new conjugate direction. The second way is to permit multiple processors to be used in a single worker process. Unlike all other parallelization steps we have taken, the use of multiple processors per worker process requires data transfer between processors in the worker environment as will be described in Section 4. In a shared memory environment, the transfer speed is fast because on-board communication channels are much faster than communication channels between nodes. In distributed memory environments, the data must be transferred between nodes using node-to-node communication channels. For this reason, we expect that the PCID execution time in a distributed memory environment will at first decrease as more processors are added to each worker process because the computational load will be shared among more processors. Eventually, the interprocessor communications will increase to the point that adding more processors to the worker process will increase execution times. This issue will be discussed further in Section 4.

Looking again at Eq.(2), we can see that one of the implications of two or more processors in a worker process means that multiple processors are available to calculate $\hat{i}_k(x)$ from $\hat{o}(x)$ and $\hat{h}_k(x)$. We take the standard approach to calculating $\hat{i}_k(x)$ by first Fourier transforming both $\hat{o}(x)$ and $\hat{h}_k(x)$, then multiplying their Fourier transforms together, and then inverse Fourier transforming the result to obtain $\hat{i}_k(x)$. We carry out the Fourier transform operation using the two-dimensional Fast Fourier Transform (FFT). Although the multiplication of the Fourier transforms can be carried out independently by the multiple processors in a worker process, the 2-D Fourier transform itself involves data transfer between the processors in a worker process. In addition, although the FFT approach generates each $\hat{i}_k(x)$ quickly on serial machines, it is the limiting factor in the amount of parallelization that can be achieved with PCID. We will discuss how we parallelize PCID to this level and the speed ramifications thereof in Section 4.

3. COMPUTATIONAL ENVIRONMENT

Until recently, we hosted the PCID software on an IBM SP/RS6000 supercomputer that is comprised of 46 Power3 nodes. Each node contains 16 Nighthawk-2 processors that run at 375 MHz and that share 8 GBytes of memory. A single processor has a theoretical peak speed of 1.5 GFLOPS. The nodes are connected via an IBM Colony Switch that has a bandwidth of ~400 MBytes/second bi-directional with an MPI latency of 17 μs. In the context of the PCID parallelization effort, the primary benefit of this supercomputer is the shared memory for up to sixteen processors. This means that a single worker process can have up to sixteen processors and still share memory, greatly reducing communication bottlenecks since the primary communication load in PCID is between processors in a single worker process. In this case, the limiting factor on speed is usually the processor speed itself. Of course, if more than sixteen processors are included in a worker process, the internode communication channel comes into play and significantly slows down PCID execution speed.

The IBM SP/RS6000 supercomputer has several limitations that have led us to procure a separate supercomputer to host PCID. The first is that the computer itself has been superseded by cheaper supercomputers that are much faster both in terms of processor speeds and communication bandwidths. The second is that the amount of RAM per processor (0.5 GBytes), significantly limits the sizes of the image arrays and number of data frames that can be included in the reconstruction process. The third reason is that this supercomputer is shared by a large number of users, making timely access to the computer problematic.

We have recently procured a Cray XD-1 supercomputer[5] to serve as a host for PCID. It is an SLES 9.0 Linux-based cluster with 144 nodes, each with two 2.4 GHz AMD64 Opteron processors and 8 GBytes of RAM. Each processor has a theoretical peak speed of ~4.8 GFLOPS. The nodes are connected via a Fat Tree RapidArray Interconnect switch that has a bandwidth of 4 GBytes/second with an MPI latency of 1.7 μs. The nodes share a Lustre high-speed parallel file system. Software on the Cray XD-1 Linux operating system includes Fortran and C compilers along with many parallel processing libraries.

The Cray XD-1 mitigates the IBM SP/RS6000 limitations mentioned previously. The Opteron processors are three times faster than the Nighthawk-2 processors. The amount of memory per processor is 4 GBytes, eight times that available on the IBM SP/RS6000. The internode communication channel is ten times faster with a latency that is ten times smaller than the IBM SP/RS6000. The main disadvantage of the Cray XD-1 is that it has only two processors per node (i.e., two processors share memory) and so the internode communication speed plays a key role in the scalability of PCID across multiple processors. This disadvantage in architecture has a corresponding monetary advantage, because the cost per GFLOP is much reduced over the cost for more specialized supercomputers. We note that, since our purchase of the Cray XD-1, commodity clusters such as the Cray XD-1 are becoming available that have more processors per node, mitigating this disadvantage.

The main computer hardware requirements for PCID to execute swiftly are a high-speed interconnect between nodes and tens to hundreds of high-speed processors. The Cray XD-1 appears to be a good computational platform for PCID with its 288 processors and high-speed interconnect. As will be shown in Section 4, our preliminary results from testing PCID on the Cray are quite promising.

4. RESULTS

The key issue in extending the parallelization of PCID to multiple processors per worker process is parallelizing the 2-D FFT across multiple processors. To carry out the FFT operation on a 2-D array, first each row is transformed using a 1-D FFT and then each column is transformed using a 1-D FFT. Because the 2-D FFT's structure involves a series of 1-D FFTs, much of the 2-D FFT is easily parallelizable. The expensive part of the 2-D FFT is transposing the array and moving array values between worker processors after the row 1-D FFTs and before the column 1-D FFTs.

For the PCID algorithm, we wrote a custom 2-D FFT routine that uses the 1-D FFT routine from the current version of FFTW (3.1.1).[6] The need to write the 2-D FFT routine is due to the lack of support for 2-D distributed-memory FFTs in FFTW 3.1.1 at this time. Despite this lack of support, we have chosen to use FFTW 3.1.1 because our tests of 1-D FFT routines to date have shown that FFTW 3.1.1 is the fastest routine available for our application. The amount of data transfer between worker processors after the row transforms and before the column transforms is minimized by having a given worker processor transfer the data it holds only to the worker processors that need the data rather than having global communications between all worker processors.

We first evaluated the execution speed of our 2-D FFT routine as a function of the number of processors used for the transform. These results were generated using the Cray XD-1 supercomputer described in Section 3. Two different array sizes were used in the test in order to explore the scalability of these transforms as a function of the amount of processing carried out by each processor and the interprocessor communications. Each timing result is the time it took to carry out a forward and inverse FFT averaged using 10^5 sequential forward and inverse FFTs. Although 10^5 is a large number of cases, we have found that even 10^4 cases is not enough to generate reliable statistics.

The FFT timing results, shown in Fig. 1, are for array sizes of $N_i = 128$ and $N_i = 512$. For the $N_i = 128$ case, notice that the computation time limits the FFT execution speed for up to 16 processors, so using up to 16 processors decreases the overall execution time. Beyond this number, the interprocessor communication times limit the execution time. For the $N_i = 512$ case, the optimum balance between computation and interprocessor communication loads occurs with 64 processors. Based just upon these two results, it appears that the optimum number of processors to use to carry out our 2-D FFT on our Cray XD-1 is one fourth to one eighth the number of rows in the array.

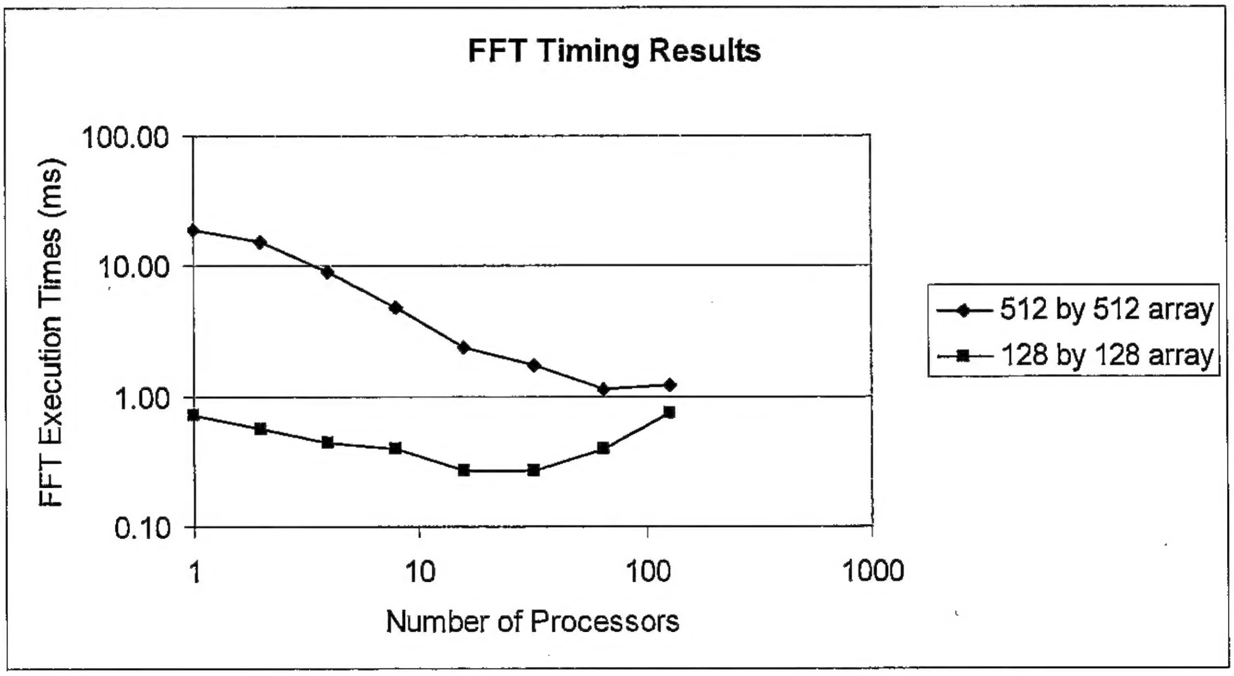

Fig. 1. FFT execution times as a function of the number of processors used.

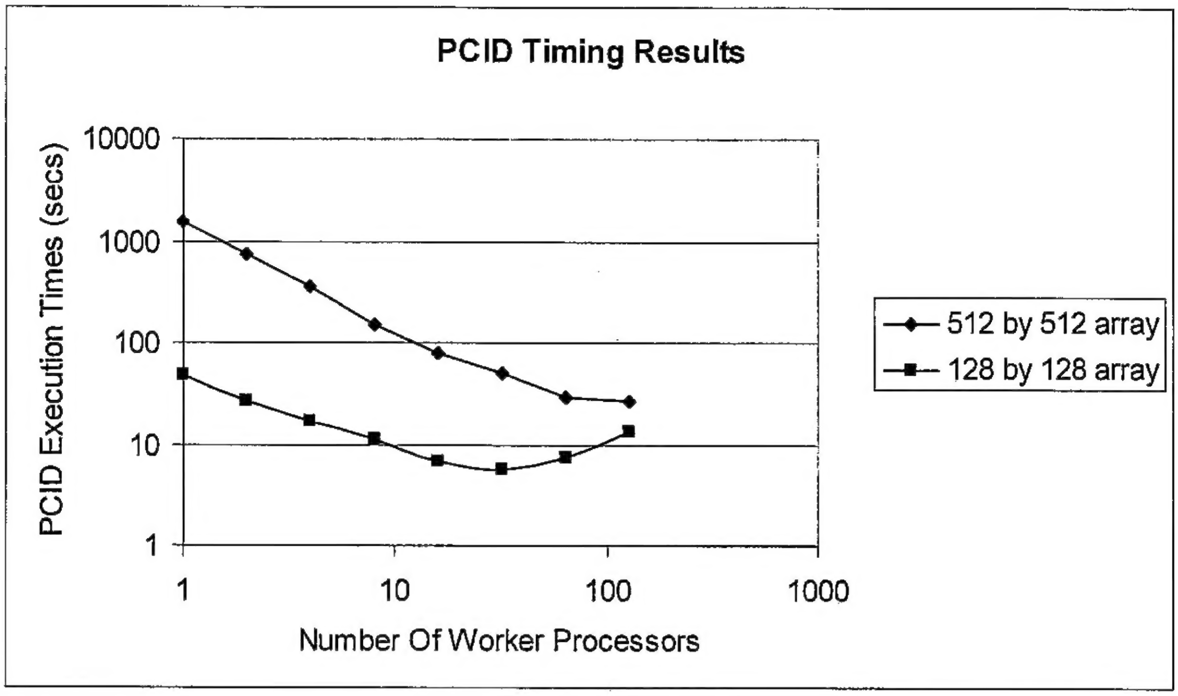

Fig. 2. PCID execution times as a function of the number of processors in a worker process.

The second set of results are the execution times of the PCID algorithm on the Cray XD-1 as a function of the number of processors included in a worker process. For these results, the PCID algorithm used our custom FFT, one blurred image as the input, one master process, one worker process, and we varied the number of processors included in the worker process. For the case of one worker processor, we hosted both the worker process and the master process on a single node. Because there are two processors per node, the master process was hosted on a separate node for two or more worker processors. Each execution time corresponds to 100 iterations. The execution times for both the $N_i = 128$ and $N_i = 512$ cases are plotted in Fig. 2. Notice that the maximum number of processors that can be used for both cases before execution times start increasing is larger than for the FFT-only results. This is expected because the PCID master process tasks each worker process to carry out more than just FFT calculations and the other calculations do not require communication between the worker processors.

The third set of results are the relative execution times of the PCID algorithm on the Cray XD-1 and the IBM/RS6000 supercomputers. For these results, we used the same parameters for the PCID algorithm as for the results in Fig. 2. In addition, for the IBM RS/6000, we hosted both the worker process and the master process on a single node as long as there were enough processors available. When the number of worker processors is sixteen or greater, the master process must be hosted on a separate node. For these sets of conditions, we calculated the execution times of PCID on both the Cray XD-1 and the IBM RS/6000 and divided the second execution time by the first to generate a ratio indicating the relative speeds of execution. The results for $N_i = 128$ case are shown in Fig. 3. Notice that, for one worker processor, the PCID execution speed on the Cray XD-1 is approximately three times faster than on the IBM/RS6000. This ratio is approximately the same ratio as the theoretical floating-point operation peak speed of the Opteron to the Nighthawk-2. The two-worker-processor case shows a decrease in the execution speed ratio that is due to the Cray XD-1 having to use two nodes while the IBM/RS6000 still only needs one node. Interestingly enough, the ratio goes down even though the two worker processors share memory. Apparently the communication load between the master and worker processors, when carried out over the internode communication channel, limits the execution speed of PCID on the Cray XD-1. The execution speed ratio stays approximately the same for the four and eight worker-processor cases, but then starts increasing significantly for sixteen or greater worker processors. This is due to the fact that the IBM/RS6000 has to use two nodes for sixteen or greater worker processors and its internode communications dominate execution speed.

The case when $N_i = 512$ is shown in Fig. 4. The conclusions for this case are qualitatively the same as for the $N_i = 128$ case except that the execution speed ratio starts increasing when there are eight worker processors. For the $N_i = 128$

case, the execution ratio starts increasing for the sixteen worker processor case due to the IBM/RS6000 computer needing to use two nodes for more than eight worker processors. The eight-worker-processor case for $N_i = 512$ still has the IBM RS/6000 computer using one node; apparently the fact that the Cray XD-1 has faster processors makes a difference even when the entire PCID executables can fit on one IBM/RS6000 node.

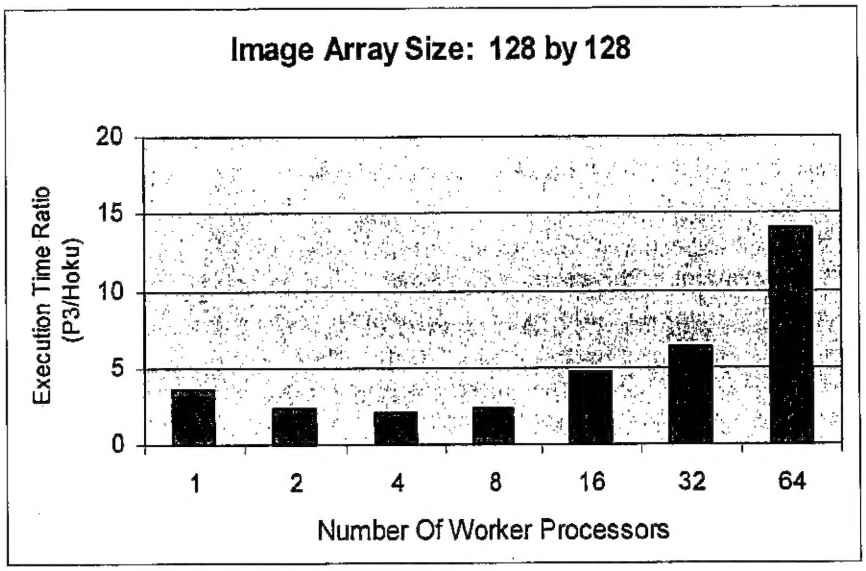

Fig. 3. The ratio of the PCID execution times on the IBM/RS6000 and the Cray XD-1 as a function of the number of processors in a worker process for $N_i = 128$.

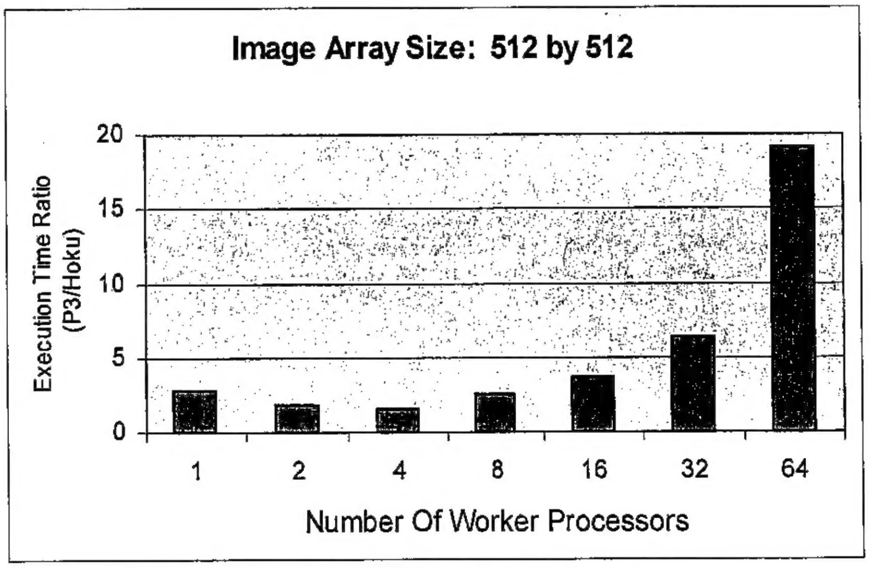

Fig. 4. The ratio of the PCID execution times on the IBM/RS6000 and the Cray XD-1 as a function of the number of processors in a worker process for $N_i = 512$.

5. CONCLUSIONS AND FURTHER WORK

We have reported on our progress in parallelizing the PCID algorithm in a scalable and portable fashion. We have structured the parallelization in such a way that the software is easily compiled and run on both shared-memory and distributed-memory platforms. By using MPICH as the parallel processing environment and not including hardware-specific instructions, PCID is essentially platform independent. The current level of parallelization permits a single worker processor to have as little as one row of one image in local memory. To illustrate the impact of this level of parallelization, consider an imaging scenario where a single 512 by 512 deblurred image is to be generated from a sequence of 100 blurred images, a not-uncommon scenario. At the current level of PCID parallelization, over 50,000 processors can be utilized by the PCID algorithm. Of course, interprocessor communications will significantly limit PCID's execution speed for this case, but this level of parallelization ensures that the execution speed of PCID will be limited by the intercommunication speed, not by the degree of parallelization of PCID. This fact is a key benefit of this level of PCID parallelization. For the first time, computational hardware is the factor limiting PCID's execution speed, not the level of PCID's parallelization.

Based on the results presented in Section 4, it appears that PCID scales well across multiple nodes. The scalability of PCID is a function of the dimensions of the images, with greater scalability for larger images.

We have carried out a first-level profiling of PCID as executed on the Cray XD-1 and have discovered that the 2-D FFT routine takes up almost 70% of the total execution time. Only 20% of that time is due to the 1-D Fourier transform operations; the remaining 50% is due to the array transpose operation and data transfer between worker processors in a worker process. Clearly, this is an area where speed improvements will have a great impact. Since the results presented herein, we have modified the array transpose and data transfer code that, in initial testing, decreases the array transpose and data transfer execution times by 30%. Translated to the overall PCID execution time, we should see a 15% reduction. We plan to include this modification in PCID in the near future.

In addition to the parallelization work reported here, we have also removed the common-block structure of the conjugate gradient minimization routines (present in the F77 Numerical Recipes version of this software) in PCID because this structure unnecessarily occupied significant amounts of memory. We plan to investigate alternate minimization routines with an eye for those that are optimized for a distributed-memory environment.

ACKNOWLEDGEMENTS

The authors thank the Air Force Research Laboratory's Directed Energy directorate and the High-Performance Computing Modernization Office for their financial support that made this research possible. The authors gratefully acknowledge the contributions of insight and beta testing from Stuart Jefferies, E. Keith Hege, and Michael Lloyd-Hart.

REFERENCES

[1] D. Kundur and D. Hatzinakos, "Blind image deconvolution," *IEEE Signal Processing Magazine*, 43-63 (May 1996).
[2] J. W. Goodman, *Introduction to Fourier Optics, 3rd ed.*, Roberts and Company Publishers, Greenwood Village (2005).
[3] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes*, Cambridge Press, Cambridge (1992).
[4] http://www-unix.mcs.anl.gov/mpi/mpich/
[5] http://www.cray.com/products/xd1/index.html
[6] http://www.fftw.org/